1、 编写算法，判断图中是否有通过给定顶点 Vk 的简单回路。

```
/* 邻接矩阵表示的无向图 */
Status judgeSimpleCycle(MGraph G, int vk){
    for(i=0; i<G. vexnum; i++)
        visited[i]=FALSE;
    /* 以 vk 为起点深度优先搜索 G，查找是否有到 vk 的简单环 */
    return DFSCycle(G,vk,vk, 0);
}
Status    DFSCycle(MGraph G, int i, int vk, int length){
    /* vk 为所给定的顶点，i 表示当前的深度搜索的起点，length 表示当前的路径长度*/
    if( i>=G.vexnum ) return FALSE;
    visited[i]=TRUE;
    for(j=0; j<G. vexnum; j++){
        /* 第i个点到第vk个顶点存在边且已有长度大于 1(排除单边情况)，则有简单回路 */
        /* 对于有向图，”&& length>1”省去 */
        if( G. arcs[i][vk].adj && length>1 ) return TRUE;
        else if( G. arcs[i][j].adj && !visited[j]){
            flag = DFSCycle( G, j, vk);
            if (flag ) return TRUE;
        }
    }
    /* 以第 i 个顶点的各邻接点*/
    return FALSE;
}
```

2、 编写算法，输出无权图中顶点v到其余每个顶点的最短路径。

```
/* 本算法基于图的层次遍历，此处为邻接矩阵表示的图 */
typedef struct FNode{
    int vexnum,parent;      /* 定义队列的元素信息，包含顶点及其父顶点的编号 */
}FNode;
void printShortPath(MGraph G, int v0){
    FNode aPath[MAX_VERTEX_NUM]; /* 依次存放所遍历的顶点，充当层次遍历的队列 */

    /* 引入 front 和 rear 分别指示队头元素在 aPath 中的下标和队尾的下一个的位置
       队列初始化      */
    front=0, rear=0;

    /* 访问标志数组初始化 */
    for(i=0; i<G.vexnum; i++)
        visited[i]=FALSE;

    /* 访问起点 */
    visited[v0]=TRUE;
    aPath[rear].vexnum = v0;        /* 保存当前访问的顶点编号*/
    aPath[rear].parent = -1;     /* 保存父顶点编号      */
    rear ++;       /* 入队 */
    while(front < rear){                /* 判断队列是否不为空 */
      /* 出队 */
        v = aPath[front++].vexnum;
        for(w=0; w<G.vexnum; w++)
            if( G.arcs[v][w].adj && !visited[w]){
                visited[w]= TRUE;
                aPath[rear].vexnum = w; aPath[rear++].parent = v;
                printPath(G, aPath, rear);
            }
    }
}

void printPath(MGraph G, FNode *path, int rear){
    i=rear-1;
    while (i>0 && path[i].parent > -1){
        printf(G.vexs[path[i].vexnum]);      /* 输出顶点信息 */
        parent = path[i].parent;
        while(path[i].vexnum!=parent)i--;     /* 找上一个输出顶点的父顶点 */
    }
    if(path[i].parent==-1)printf(G.vexs[path[i].vexnum]);
}
```

3、 编写算法，判断无权图中，顶点V_1和V_2之间是否存在一条长度为k的简单路径。

```
Status judgeSimplePath(MGraph G, int v1, int v2, int k){
    for(i=0; i<G.vexnum; i++)
        visited[i]=FALSE;
    return DFSPath(G,v1,v2,k,0);   /* 最后一个参数表示当前的路径长度 */
}
Status    DFSPath(MGraph G, int i, int v2, int k, int length){
    if( i>=G.vexnum || length>k ) return FALSE;
    if (i==v2)
        if (length==k) return TRUE;
        else if (length<k)          return FALSE;
    visited[i]=TRUE;
    for(j=0; j<G.vexnum; j++){
        if( G.arcs[i][j].adj && !visited[j]){
            flag = DFSPath( G, j, v2, k,length+1);
            if (flag ) return TRUE;
        }
    }
    return FALSE;
}
```

4、 编写算法，统计无权图中，距v0最短路径长度最长的顶点个数。

```
typedef struct LvlNode{
    int vexnum,level;
}LvlNode;
int countSPathLong(MGraph G, int v0){
    LvlNode aVexs[MAX_VERTEX_NUM]; /*  依次存放所访问顶点  */

    /*  队列初始化      */
    front=0, rear=0;

    /*  访问标志数组初始化  */
    for(i=0; i<G.vexnum; i++)
        visited[i]=FALSE;

    /*  访问起点   */
    visited[v0]=TRUE;
    aVexs[rear].vexnum = v0;    /*  保存当前访问的顶点编号*/
    aVexs[rear].level = 1;          /*  保存顶点的层号      */
    rear ++;        /*  入队   */
    while(front < rear){                    /*  判断队列是否不为空  */
        /*  出队  */
        v = aVexs[front].vexnum; i = aVexs[front++].level;
        for(w=0; w<G.vexnum; w++){
            if( G.arcs[v][w].adj && !visited[w]){
                visited[w]= TRUE;
                aVexs[rear].vexnum = w; aVexs[rear++].level = i+1;
            }
        }
    }
    i = rear-1;
    while( i>=0 && aVexs[i].level == aVexs[rear-1].level){
        printf("%c,", G.vexs[aVexs[i].vexnum]);
        i--;
    }
    return rear-i-1;
}
```

5、 编写算法，输出距离无权图中顶点*v*的最短路径长度为*k*的所有顶点。

```
typedef struct LvlNode{
    int vexnum,level;
}LvlNode;
void printSPathK(MGraph G, int v0, int k){
    LvlNode aVexs[MAX_VERTEX_NUM]; /*  存放当前层的顶点  */

    /*  队列初始化      */
    front=0, rear=0;

    /*  访问标志数组初始化  */
    for(i=0; i<G.vexnum; i++)
        visited[i]=FALSE;

    /*  访问起点   */
    visited[v0]=TRUE;
    aVexs[rear].vexnum = v0;    /*  保存当前访问的顶点编号*/
    aVexs[rear].level = 0;          /*  保存当前的路径长度      */
    rear ++;        /*  入队   */
    while(front < rear){                    /*  判断队列是否不为空  */
      /*  出队  */
        v = aVexs[front].vexnum; i = aVexs[front++].level;
        if (i==k) return;
        for(w=0; w<G.vexnum; w++)
            if( G.arcs[v][w].adj && !visited[w]){
                if ( i==k-1)
                    printf(G.vexs[w]);
                visited[w]= TRUE;
                aVexs[rear].vexnum = w; aVexs[rear++].level = i+1;
            }
    }
}
```